CHAPTER 3

Numerical Data

In this chapter you will learn about

- Constants and variables
- Integers
- GUI Object IntTypeIn
- Real numbers
- GUI Object FloatTypeIn

C. Thomas Wu

Constant and Variable

- The simplest kind of data used in a computer program is a constant, which is a data value that remains unchanged.

- One kind of constant data we have seen in the sample programs is a *literal constant*. Let's look again at the sample program that draws a square.

```
// Program Square:  A program that draws a square.
#include "Turtles.h"
void main ( )
{
  Turtle myTurtle;
  myTurtle.Init(260,180); // Start from location 260,180.

  myTurtle.Move(50); // Draw the bottom of the square.
  myTurtle.Turn(90); // Turn to draw the right side.

  myTurtle.Move(50); // Draw the right side.
  myTurtle.Turn(90); // Turn to draw the top.

  myTurtle.Move(50); // Draw the top.
  myTurtle.Turn(90); // Turn to draw the left side.

  myTurtle.Move(50); // Draw the left side.

  myTurtle.Done();
}
```

- The numeric arguments such as 260, 180, 50, and 90 are literal constants (to be precise, they are literal integer constants). They are constants because their values do not change, and they are literal because we "literally" specify their constant data values instead of using some kinds of identifiers.

- Porgram Square always draws a square 50 pixels in size. If we to draw squares of different sizes, we can use a *named*, or *symbolic, constant*. Here's the modified program with a named constant.

```
// Program Square4: A program that draws a square
//                  whose side has the length of size.
#include "Turtles.h"
void main ( )
{
  const int size = 50;// Use the named constant size.
  Turtle myTurtle;
  myTurtle.Init(260,180); // Start from location 260,180.

  myTurtle.Move(size);  // Draw the bottom of the square.
  myTurtle.Turn(90);    // Turn to draw the right side.

  myTurtle.Move(size);  // Draw the right side.
  myTurtle.Turn(90);    // Turn to draw the top.

  myTurtle.Move(size);   // Draw the top.
  myTurtle.Turn(90);    // Turn to draw the left side.

  myTurtle.Move(size);   // Draw the left side.

  myTurtle.Done();
}
```

- The statement

```
const int size = 50;
```

  declares that `size` is an integer constant whose value is 50. Notice the reserved words `const` and `int` for declaring a constant data value of type integer.

- The named constant `size` is used in four different places in the program. By using this named constant, whenever we want to run the program again with another value for the size of a square, the only statement we have to modify is the constant declaration statement.

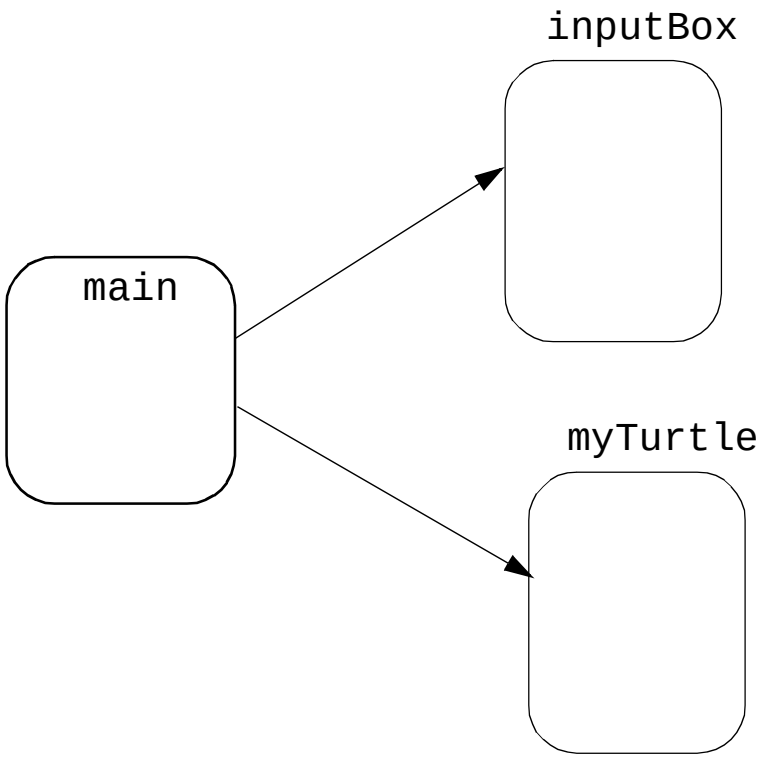
- Although the use of a named constant improved the situation somewhat, drawing many squares with different size is still a chore. Imagine trying out squares of different sizes until you find the one you like. You may end up going through the edit-compile-run cycle 20 or 30 times.

- A program that requires us to edit and compile again for each new value is not very useful. To make the program more useful, we need an input statement that gets a value for the size of a square.

- Here is an improved program that uses a new GUIobject called `IntTypeIn` for getting a value.

```
// Program AnySquare: A program that draws a square of
//                    any size.

#include "Turtles.h"
#include "GUIObj.h"

void main ( )
{
  int         size;
  IntTypeIn   inputBox;// object for getting input
  Turtle      myTurtle;

  myTurtle.Init(260,180); // Start from location 260,180.

  //Get the square size from the user.
  size = inputBox.GetInt("Square Size",
              "Enter the square size");

  myTurtle.Move(size);  // Draw the bottom of the square.
  myTurtle.Turn(90);    // Turn to draw the right side.

  myTurtle.Move(size);  // Draw the right side.
  myTurtle.Turn(90);    // Turn to draw the top.

  myTurtle.Move(size); // Draw the top.
  myTurtle.Turn(90);    // Turn to draw the left side.

  myTurtle.Move(size); // Draw the left side.

  myTurtle.Done();

}
```

- Here is the object diagram for the above program.



- The identifier `size` is not a constant anymore; now it is a *variable*. The declaration statement

```
int size;
```

  declares that `size` is the name for a variable of type `int` (integer).

- Instead of saying "a size is the name for a variable of integer type," we say "a size is an integer variable."

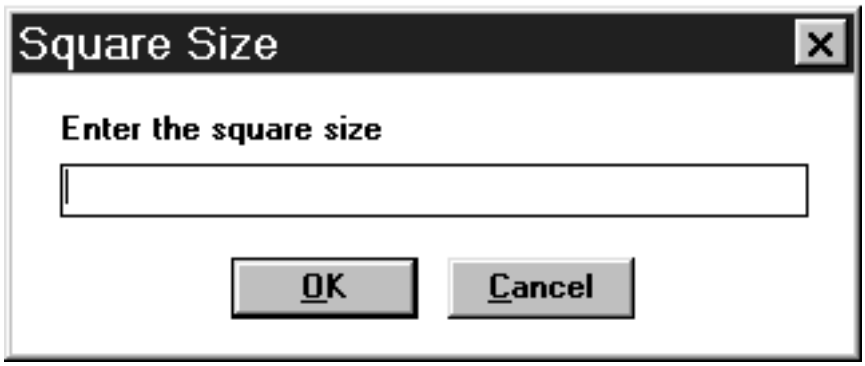
- A variable is very similar to a constant in many ways (notice that the calls to the `Move` functions are the same in both programs) except for one big difference.

- A constant value remains fixed, but a value assigned to a variable can vary, thus the name *variable*. A value assigned to a variable can be of any value of a declared type. In the above program, for example, the identifier `size` is declared to be an integer variable.

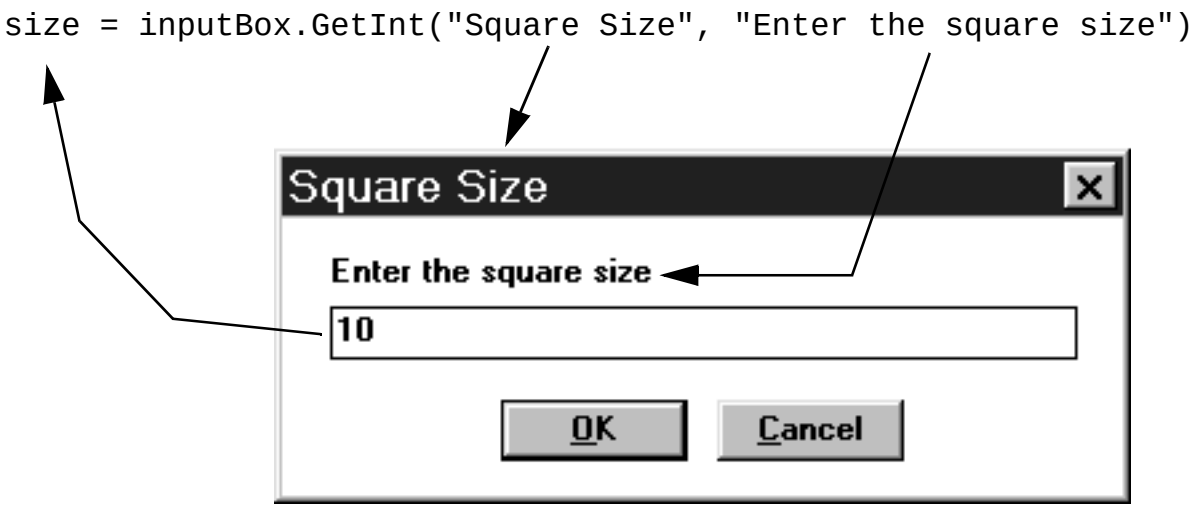
- Every time the AnySquare program is executed, it gets a new value from the user via the `IntTypeIn` object `inputBox`. When the function `GetInt` of `inputBox` is invoked by the statement

```
inputBox.GetInt("Square Size", "Enter the square size");
```

the following message box appears on the screen.



- The following diagram summarizes how the `GetInt` function works.



- When you click the Cancel button, the entered value is erased and `inputBox` remains on the screen. When you enter invalid data (a noninteger value or no value at all) and click the OK button, an error message is displayed. Again, `inputBox` remains on the screen. The only way to close an `IntTypeIn` object, that is, to make it disappear from the screen, is to enter a correct value and click the OK button.

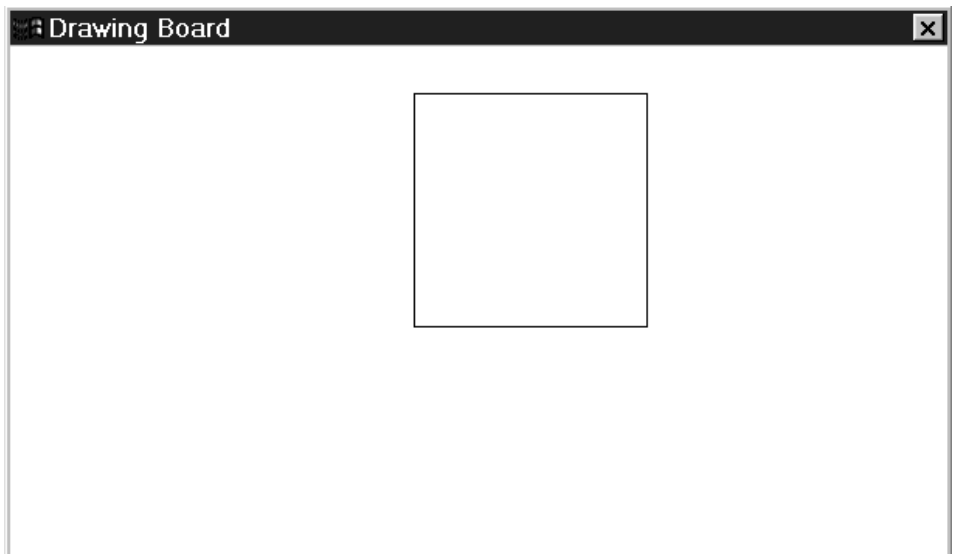
- A C++ function is very much like an ordinary mathematical function because it returns the result from a given number of arguments. However, a C++ function is not limited to returning numerical values—it can return any valid C++ data value, including objects. A C++ function does not even have to return a value.

- An integer value entered by the user is assigned to the variable `size`, and then, this value is passed to the function `Move` in the statement

```
myTurtle.Move(size)
```

- If the user enters `10` and clicks the OK button, the square would be

- If the number `150` is entered instead, then the square would be



- By incorporating an input routine and using a variable, we made a significant improvement to the program. With this improved program, we do not have to modify the program to draw squares of different sizes. Editing and compiling the program for each new value of a square size is not necessary any more. We only need to execute the program to draw a square of different size.

C. Thomas Wu

Integers

- We declare a variable by designating its name (identifier) and type, and we declare a constant by designating its name, type, and value. In declaring a variable, we can also assign an initial value to it.

- Let's look at examples.

```
int         x = 435;
const int   v = 123;
long        y = 25310, z, w;
```

The first declaration declares the identifier `x` as an integer variable with the initial value of `435`. Since `x` is a variable, its value can change later in the program.

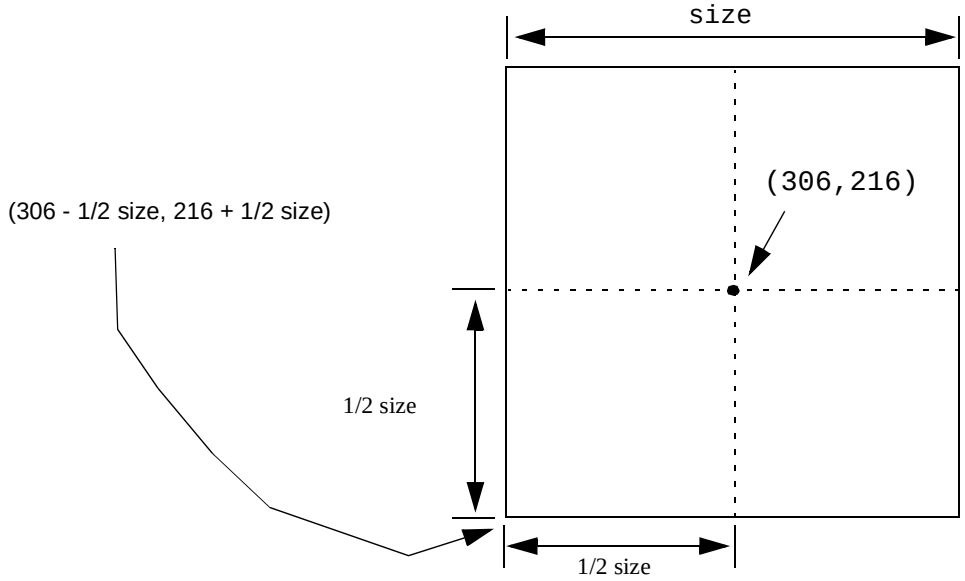
The second declaration declares the identifier `v` as an integer constant whose value is `123`.

The last declaration declares three integer variables `y`, `z`, and `w`. The variable `y` is also initialized to the value `25310`. The type `long` (or equivalently `long int`) is used to represent a larger integer value.

- The following table describes the various data types for storing integers.

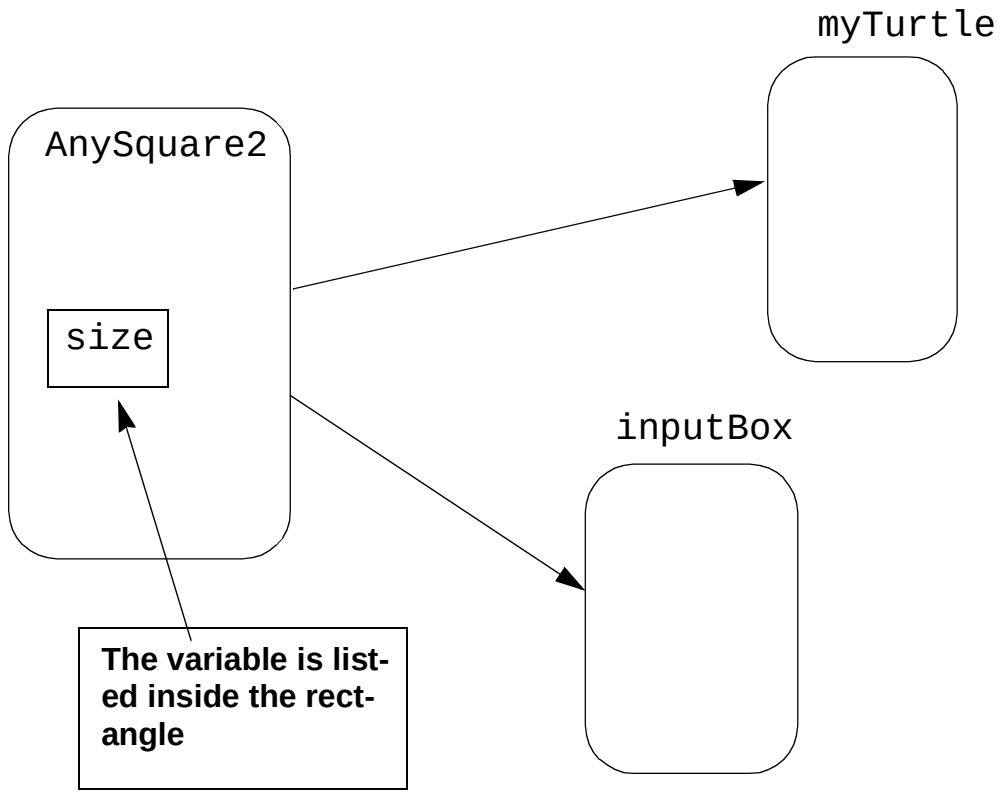
Data Type	Explanation
int	An integer value ranging from -32,768 to 32,767. Uses 2 bytes.
short	Shorthand for short int. Normally equivalent to int, but it could be different depending on the actual C++ compiler.
long	Shorthand for long int. An integer value ranging from -2,147,483,648 to 2,147,483,647. Uses 4 bytes.

- The following example illustrates an arithmetic computation involving integer values. Suppose we want to draw a square of a given size and position it at the center of a window. To do so, the program must first move the turtle to the correct position before drawing the square as shown in the illustration below.



- The following program draws a square at the center of the window using the above expression.

```
// Program AnySquare2: A program that draws a square of any
//                     size at the center of the window.
#include "Turtles.h"
#include "GUIObj.h"
void main ( )
{
  int           size;   // size of a square
  IntTypeIn     inputBox;
  Turtle        myTurtle;

  //Get the size from the user
  size = inputBox.GetInt("Square Size",
         "Enter the square size");

  myTurtle.Init(306 - size/2, 216 + size/2);

  // Initialize a turtle to correct
  // starting point so the square is centered.
  myTurtle.Move(size);  // Draw the bottom of the square.
  myTurtle.Turn(90);    // Turn to draw the right side.
  myTurtle.Move(size);  // Draw the right side.
  myTurtle.Turn(90);    // Turn to draw the top.
  myTurtle.Move(size); // Draw the top.
  myTurtle.Turn(90);    // Turn to draw the left side.
  myTurtle.Move(size); // Draw the left side.
  myTurtle.Done();
}
```



- The C++ arithmetic operators for integer operands are shown in the following table. The examples assume that `x` and `y` are integer variables with values `10` and `3`, respectively.

Operator	Description	Example Expression	Expression Value (x=10; y=3)
+	addition	x + y	13
-	subtraction	x - y	7
*	multiplication	x * y	30
/	quotient division (integer)	x / y	3
%	modulo division -- remainder after division	x % y	1
-	unary minus	- y	- 3
+	unary plus	+ y	3

- The rules to combine operators and operands to formulate valid C++ arithmetic expressions (limited to integer operands) are

    - Operands may be integer constants (e.g., `12`, `77`, `1`), integer variables (e.g., `i`, `size`, `n`) or other arithmetic expressions possibly enclosed in parentheses (e.g., `(size + 5)`, `(n – 2)`).

    - Each binary operator (`=`, `–`, `*`, `/`, `%`) must have an operand on either side. It is called a binary operator because it requires two operands.

    - A unary operator must have an operand to its right. Thus `–3` is a valid arithmetic expression but `3–` is not.

    - Parentheses must match—left parentheses must have matching right parentheses. For example, `(3 + 5)` is valid while `( 3 + ( 5 – 3)` is not.

- C++ uses an assignment operator for assigning a value to a variable. The equal symbol is used for an assigment operator, and the general format is

    `variable = value`

    where `value` is any expression that results in a value when evaluated. Here are some examples:

```
x = 2;
y = x + 3;
z = x * y - 4;
```

When the three statements are executed, the variables `x`, `y`, and `z` will have values `2`, `5`, and `6`, respectively.

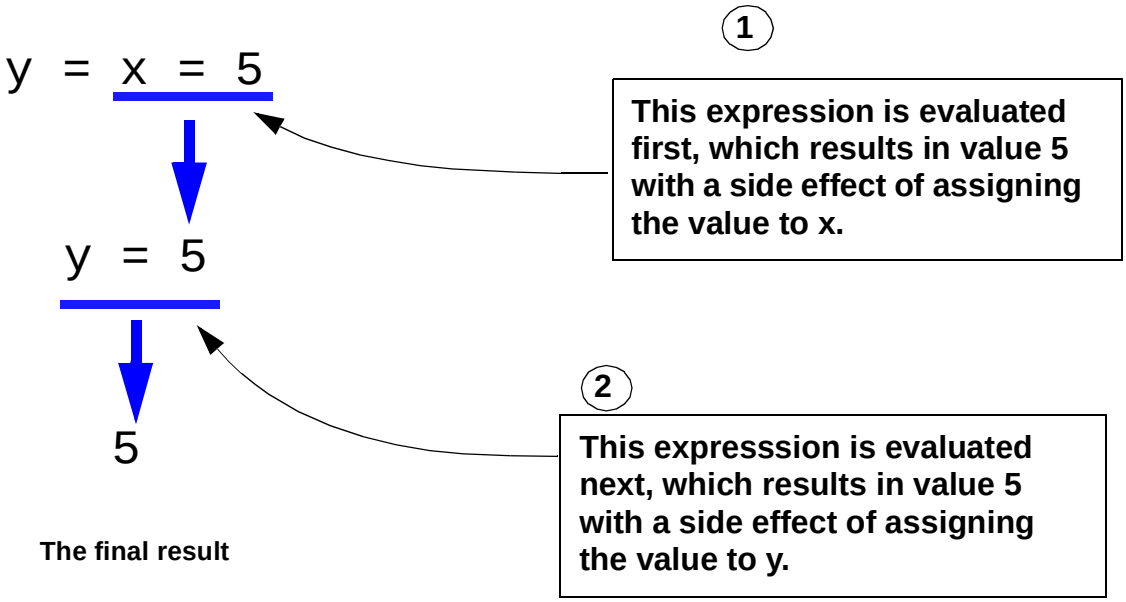
- If we want to set the variables `x`, `y`, and `z` to `5`, instead of stating

```
x = 5;
y = 5;
z = 5;
```

we could state

```
z = y = x = 5;
```

- This is how we interpret cascaded assignment operators:



Side effect: Value 5 is assigned to x and y

- When an arithmetic expression has more than one operator, the result depends on the order in which the operators are evaluated. For example, the expression

```
5 + 2 * 4
```

  could be evaluated to `28` or to `13`, depending on whether the addition or the multiplication is performed first.

- C++ defines an *operator precedence*, which determines the order of evaluating expressions, so an expression is evaluated precisely in one way.

- C++ follows these operator precedence:

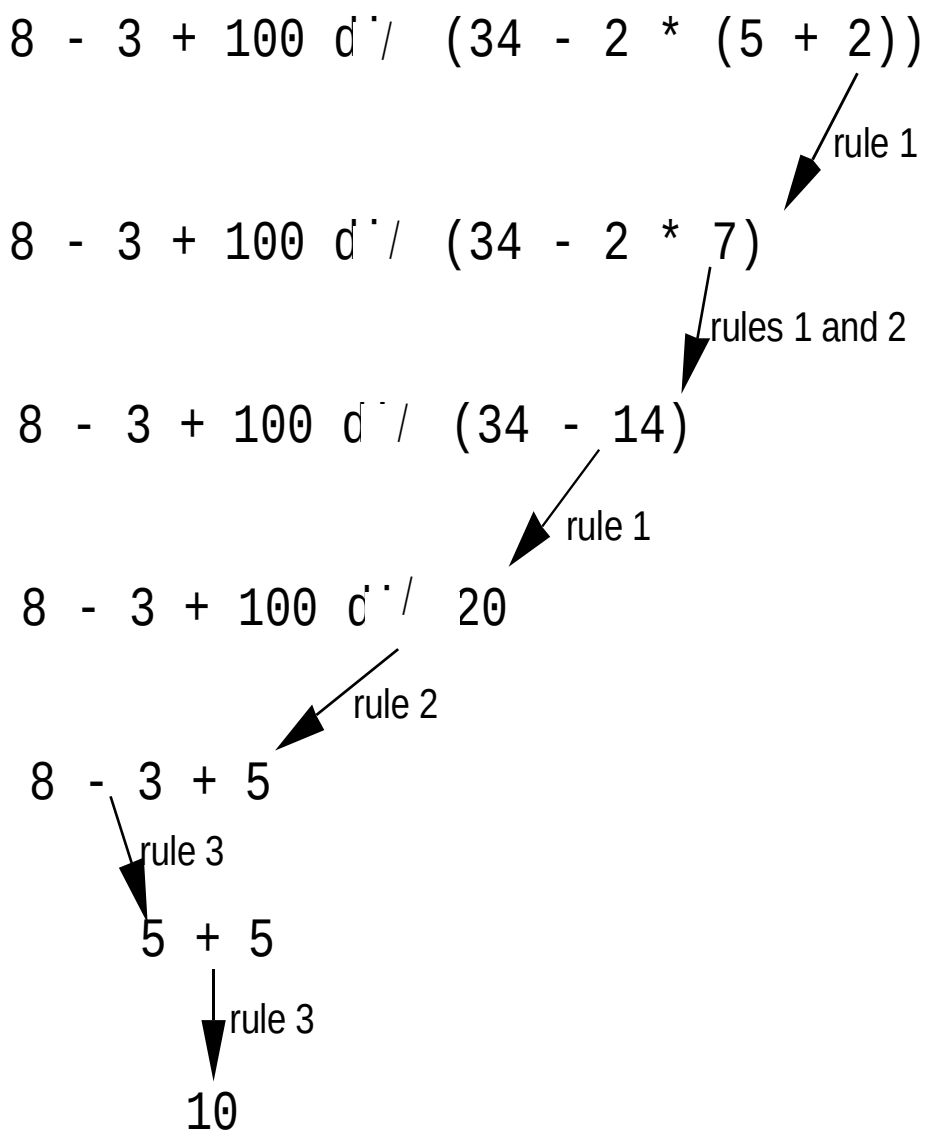
    - Subexpressions enclosed in parentheses are evaluated before operators outside the parentheses are evaluated.

    - The operators are grouped into five categories. The highest category contains the unary operators. The second category has multiplicative operators (`*`, `/`, and `%`), the third category has additive operators (`+` and `-`), and the last category has assignment operators. Operators in the higher categories are applied before those in a lower category unless parentheses dictate otherwise (see rule 1).

    - In the absence of parentheses, if two operators are from the same category, then the operators are evaluated left to right except for the assignment and unary operators, which are evaluated right to left.

C. Thomas Wu

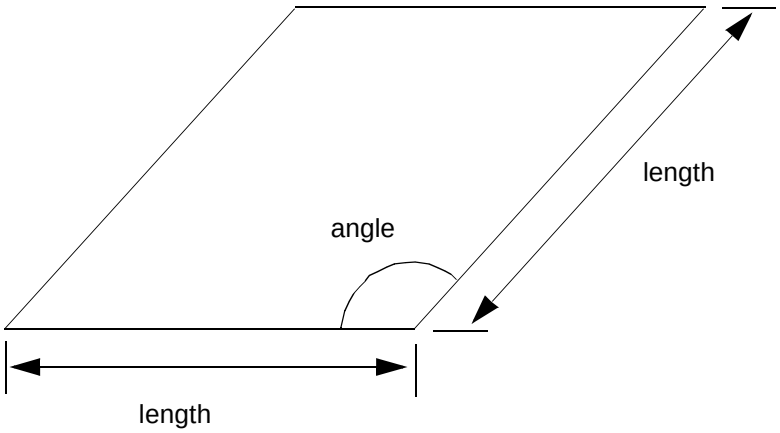
- The following example illustrates the precedence rules.



- We must know the precedence rules in order to read other programmers' code. However, when we write an arithmetic expression ourselves, we should use parentheses and not rely on the precedence rules. The use of parentheses makes the order of evaluation explicit, so it eliminates the needs to remember the precedence rules in detail.

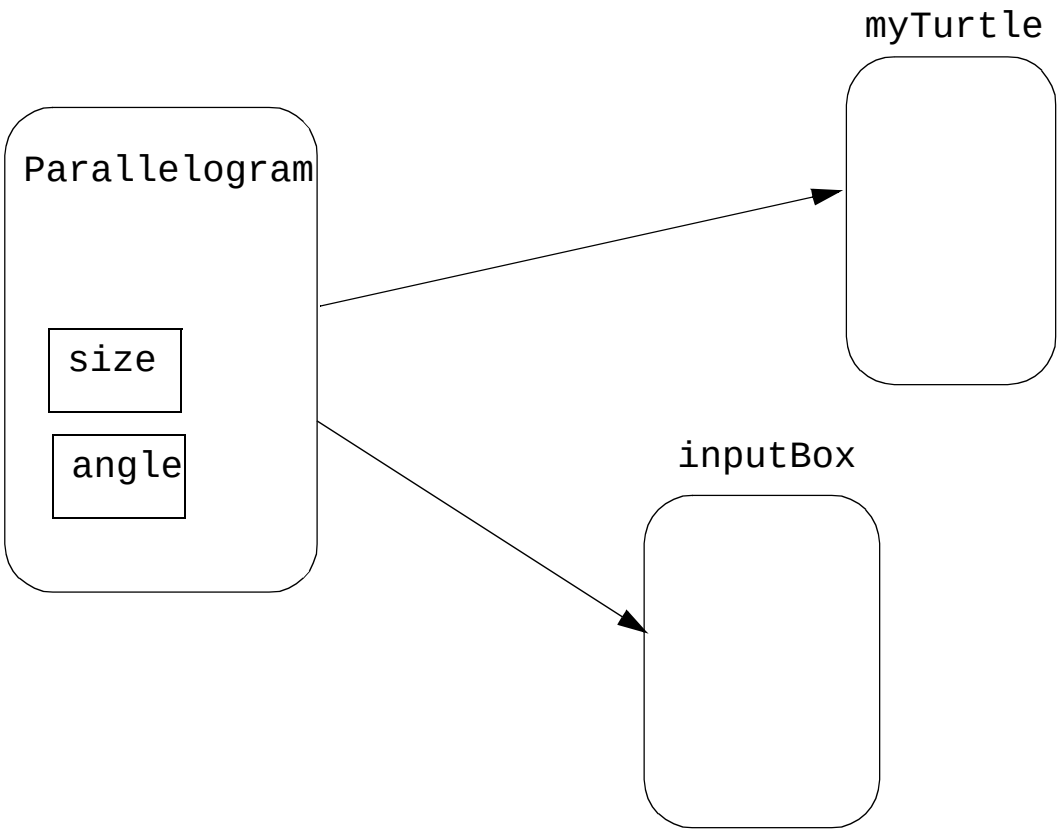
Sample Programs Using Integers

- Let's write a program that draws an equilateral parallelogram given values for the length of the sides and the angle between the base and the right side.

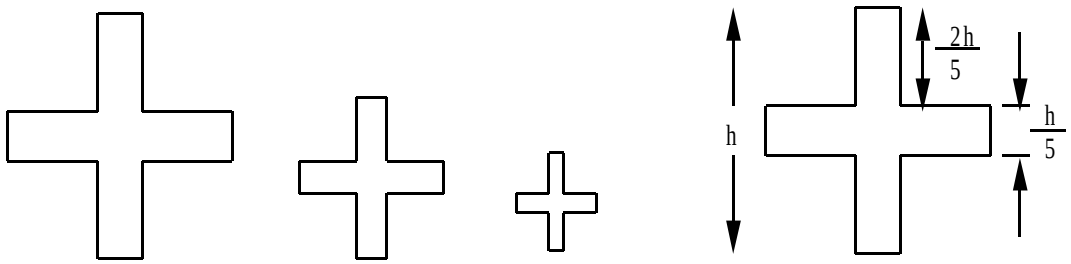



```
// Program Parallelogram: A program that draws a
//                        parallelogram of length and angle.
#include "Turtles.h"
#include "GUIObj.h"
void main ( )
{
  int         length, angle;
  IntTypeIn   inputBox;
  Turtle      myTurtle;

  myTurtle.Init(306,216); // Start from the center.

  //Input the data from the user
  length = inputBox.GetInt("Length", "Enter the length");
  angle  = inputBox.GetInt("Angle", "Enter the angle");
```

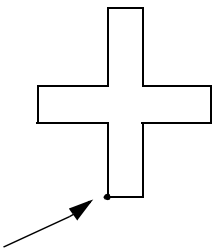
```
   myTurtle.Move(length);  // Draw the bottom.
   myTurtle.Turn(180 - angle); // Turn to draw the right side.

   myTurtle.Move(length);  // Draw the right side.
   myTurtle.Turn(angle);  // Turn to draw the top.

   myTurtle.Move(length); // Draw the top.
   myTurtle.Turn(180 - angle);  // Turn to draw the left side.

   myTurtle.Move(length); // Draw the left side.

   myTurtle.Done();
}
```



- The next sample program will draw a plus sign of any size. We'll assume that the width of each of the four legs of the plus sign should be one-fifth of the height. Therefore the length of each leg would be two-fifths of the height.

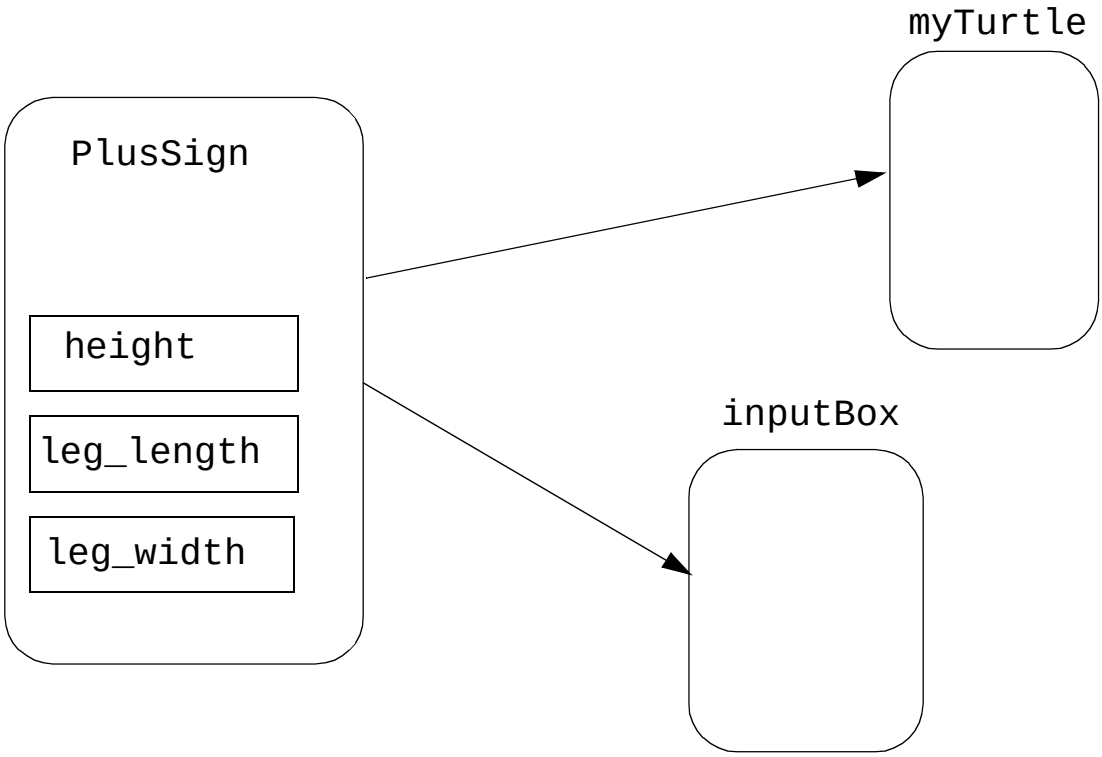



```
// Program PlusSign: A program that draws a plus sign.

#include "Turtles.h"
#include "GUIObj.h"

void main ( )
{
  int height, leg_length, leg_width;
  IntTypeIn inputBox;
  Turtle    myTurtle;

  myTurtle.Init(306,216); //starting point

  //Get values from the user.
  height = inputBox.GetInt("Length","Enter the height");

  leg_width = height / 5;
  leg_length = 2 * leg_width;

  myTurtle.Move(leg_width);  // End of bottom leg
  myTurtle.Turn(90);
```



```
myTurtle.Move(leg_length);  // Right side of bottom leg
myTurtle.Turn(-90);

myTurtle.Move(leg_length);  // Bottom of right leg
myTurtle.Turn(90);

myTurtle.Move(leg_width);   // End of right leg
myTurtle.Turn(90);

myTurtle.Move(leg_length);  //Top of right leg
myTurtle.Turn(-90);

myTurtle.Move(leg_length);  //Right side of top leg
myTurtle.Turn(90);

myTurtle.Move(leg_width);   //End of top leg
myTurtle.Turn(90);

myTurtle.Move(leg_length);  //Left side of top leg
myTurtle.Turn(-90);

myTurtle.Move(leg_length);  //Top of left leg
myTurtle.Turn(90);

myTurtle.Move(leg_width);   //End of left leg
myTurtle.Turn(90);

myTurtle.Move(leg_length);  //Bottom of left leg
myTurtle.Turn(-90);
```

```
   myTurtle.Move(leg_length);  //Left side of bottom leg
   myTurtle.Turn(90);

   myTurtle.Done();
}
```



Real Numbers

- In C++, we have data types `float`, `double`, and `long double` for representing real numbers. The word `float` is derived from the way the real numbers are stored in a computer, that is, as a floating-point representation.

- The actual number of bytes used for representing different precisions depends on the computer or compiler being used. It is typically 4 bytes for `float`, 8 bytes for `double`, and 10 bytes for `long double`. The following describes the three data types for representing real numbers. Notice that the table lists the information most representative of all compilers; your compiler may have slightly different precisions.

Data Type	Explanation
float	A real value with 7-digit precision ranging from 3.4×10^{-38} to 3.4×10^{38}. Uses 4 bytes.
double	A real value with 15-digit precision ranging from 1.7×10^{-308} to 1.7×10^{308}. Uses 8 bytes.
long double	A real value with 19-digit precision ranging from 3.4×10^{-4932} to 1.1×10^{4932}. Uses 10 bytes.

- The result of division between two integers is integer (with a fractional part truncated; e.g. 6 / 4 is 1), and the result of division between two real numbers is real (e.g., 6.0 / 4.0 is 1.5).

- What about a case when one of the operands is an integer and another a real? The result is a real. In other words, the integer is first converted to a real and then the operation is carried. This is because the `float` has a higher precision than an `int`:

converting an `int` to a `float` will not result in any loss of data, but converting a `float` to an `int` could.

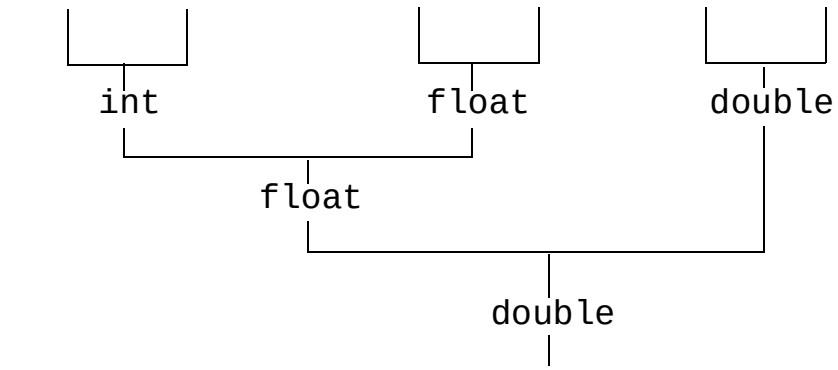
- The following statements cause the variable `y` to hold a `float` value `1.5`.

```
int i = 6;
float x = 4.0, y;

y = i / x;
```

- An arithmetic expression that contains different data types is called a*mixed-mode expression*. The mixed-mode expression is evaluated by converting the data types of operands so the operations are carried out between the same data types.

- In mixed-mode expressions, a lower-precision data value is converted to the next higher precision data type. The ranking of data types from high to low is `long double`, `double`, `float`, `long int`, `int`, and `short`.

```
int    i;
float  f;
double d;

( i  /  5 )  +   ( f  +  i ) - ( d  *  i )
```



■

A Sample Program Using Real Numbers

- As an illustration of using real numbers, we will write a program to solve quadratic equations of the form

$$Ax^2 + Bx + C = 0$$

where the coefficients A, B, and C are real numbers. The two solutions are derived by the formula

$$x = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}$$

- We assume that

$$B^2 \geq 4AC$$

is true, so there will be either one or two real number solutions for `x`. Here is the program.

```
//Program Quad: Find the two solutions for the quadratic
//        equation Ax^2 + Bx + C = 0.

#include "GUIObj.h"
#include "math.h"

void main ()
{
```

```
   float A, B, C, x1, x2, sqrtOfDiscriminant;
   FloatTypeInfloatInpBox;
   OKBox  msgBox;

   //get three inputs
   A = floatInpBox.GetFloat("INPUT","Type in value for A");
   B = floatInpBox.GetFloat("INPUT","Type in value for B");
   C = floatInpBox.GetFloat("INPUT","Type in value for C");

   //compute the sqrt of a discriminant and two solutions
   sqrtOfDiscriminant = sqrt( B*B - 4*A*C );
   x1 = (-B + sqrtOfDiscriminant) / (2 * A);
   x2 = (-B - sqrtOfDiscriminant) / (2 * A);

   //display the results
   msgBox.Display(x1);
   msgBox.Display(x2);
}
```

- The program uses one new GUI object `FloatTypeIn` for inputting values of `float` data type. The function to accept an input value is `GetFloat`, whose functionality is analogous to the `GetInt` function of `IntTypeIn`. The only difference is whether an input value is an integer or a real number. For both functions the first argument is the title of the input box, and the second argument is the prompt inside the input box.

- The function `sqrt`, which is used to compute the discriminant, is a compiler-supplied function defined in the header file `math.h`.